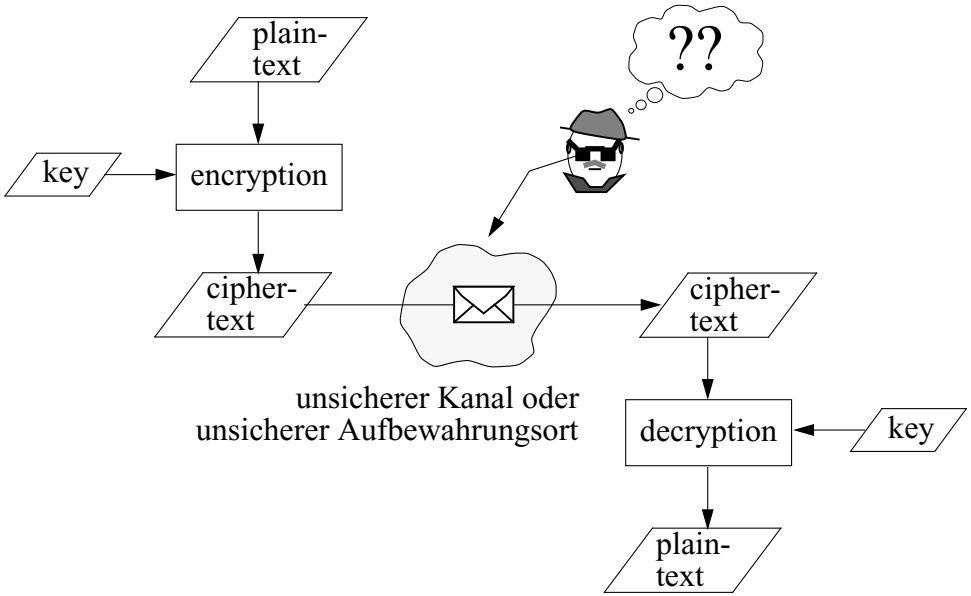
Kryptosysteme



- Schreibweisen
  - *Verschlüsseln* mit Schlüssel K_1: Schlüsseltext = { Klartext }$_{K1}$
  - *Entschlüsseln* mit Schlüssel K_2: Klartext = { Schlüsseltext }$_{K2}$
- *Symmetrische* Kryptosysteme: K1 = K2
- A*symmetrische* Kryptosysteme: K1 ≠ K2

Kryptosysteme (2)

- Geheimhalten des Verschlüsselungsverfahrens stellt i.Allg. kein Sicherheitsgewinn dar!
  - organisatorisch oft nicht lange durchhaltbar
  - kein öffentliches Feedback über erkannte Schwächen des Verfahrens
  - Verfahren, die Geheimhaltung nötig hätten, erscheinen "verdächtig"
- Verschlüsselungsfunktion ist ohne Kenntnis der Schlüssel höchstens mit unverhältnismässig hohem Rechenaufwand umkehrbar

---

- Nachteile symmetrischer Schlüssel:
  - Schlüssel muss geheimgehalten werden (da Verfahren i.Allg. bekannt)
  - mit allen Kommunikationspartnern separaten Schlüssel vereinbaren
  - hohe Komplexität der Schlüsselverwaltung bei vielen Teilnehmern
  - Problem des geheimen Schlüsselaustausches
- Vorteile symmetrischer Schlüssel:
  - ca. 100 bis 1000 Mal schneller als typische asymmetrischeVerfahren
- Beispiele für symmetrische Verfahren:
  - IDEA (International Data Encryption Algorithm): 128-Bit Schlüssel, Einsatz in PGP
  - DES (Data Encryption Standard)
  - AES (Advanced Encryption Standard) als Nachfolger von DES

One-Time Pads

- “Perfektes” (symmetrisches) Kryptosystem
  - Denkübung: unter welchen Voraussetzungen?

- Prinzip: Wähle zufällige Sequenz von Schlüsselbits
  - *Verschlüsselung:* Schlüsseltext = Klartext XOR Schlüsselbitsequenz
  - *Entschlüsselung:* Klartext = Schlüsseltext XOR Schlüsselbitsequenz
  - Begründung: ((a XOR b) XOR b) = a (für alle Bitbelegungen von a, b)

Klartext	**V**	**E**	**R**	**T**	**E**	**I**	**L**	**T**	**E**		**S**	**Y**	**S**	**T**	**E**	**M**	**E**
in ASCII	56	45	52	54	45	49	4C	54	45	20	53	59	53	54	45	4D	45
									XOR								
Schlüssel	**4C**	**93**	**EF**	**20**	**B7**	**55**	**92**	**7C**	**DA**	**69**	**23**	**F8**	**BB**	**72**	**0E**	**81**	**00**
= Chiffre	**1A**	**D6**	**BD**	**74**	**F2**	**1C**	**DE**	**28**	**9F**	**49**	**70**	**A1**	**E8**	**26**	**4B**	**CC**	**45**

- Anforderungen an Schlüsselbitsequenz:
  - keine periodische Wiederholung von Bitmustern
    → Schlüssellänge = Klartextlänge
  - Schlüsselbitsequenz ohne Bildungsgesetz (“echte” Zufallsfolge)
  - Schlüsselbitsequenz ist wirklich “one-time“ (keine Mehrfachverwendung!)

- Kryptoanalyse ohne Kenntnis der Schlüsselbitsequenz ist dann nicht möglich

- Nachteile von One-Time Pads:
  - Verwendung unhandlich (hoher Bedarf an frischen Schlüsselbits, dadurch aufwändiger Schlüsselaustausch)
  - Synchronisationsproblem bei Übertragungsstörungen (wenn Empfang ausser Takt gerät ist Folgetext verloren)
  - nur für hohe Sicherheitsanforderungen gebräuchlich (z.B. “rotes Telefon”)

Pseudo-Zufallszahlen?

Security Loophole Found in Microsoft Windows

University of Haifa, 12 Nov 2007

A group of researchers in Israel notified Microsoft that they have discovered a security loophole in the Windows 2000 operating system.

The researchers say they have found a way to decipher how Windows’ random number generator works, compute previous and future encryption keys used by a computer, and monitor private communication. The security loophole jeopardizes emails, passwords, and credit card numbers entered into a computer. "This is not a theoretical discovery," says Dr. Benny Pinkas from the Department of Computer Science at the University of Haifa, who headed the research initiative. "Anyone who exploits this security loophole can definitely access this information on other computers."

The researchers say the newer versions of Windows may also be vulnerable if Microsoft uses similar random number generator programs.

Asymmetrische Kryptosysteme

Schlimm sind die Schlüssel, die nur schliessen auf, nicht zu;
Mit solchem Schlüsselbund im Haus verarmest du.
Friedrich Rückert, *Weisheit des Brahmanen*

- Schlüssel zum Ver- / Entschlüsseln sind *verschieden*
  - *RSA-Verfahren* (Rivest, Shamir, Adleman, 1978), beruht auf der Schwierigkeit von Faktorisierung
  - andere Verfahren beruhen z.B. auf diskreten Logarithmen

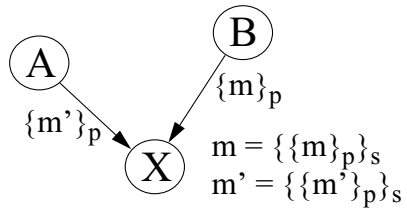
- Für jeden Prozess X existiert ein Paar (p,s)

  p = *public key* ← zum *Verschlüsseln* von Nachrichten an X

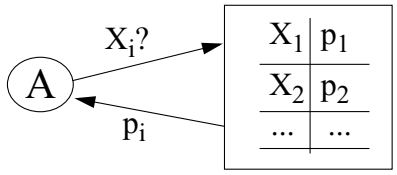
  s = *secret key* (oder "private" key) ← zum *Entschlüsseln* von mit p verschlüsselten Nachrichten

- Jeder Prozess, der an X sendet, kennt p

- Nur X selbst kennt s



- *Public-Key-Server:*
  Welchen Schlüssel hat Prozess X_i?



  - Server muss vertrauenswürdig sein
  - Kommunikation zum Server darf nicht manipuliert sein
  - Vielleicht tut es auch ein öffentliches "Telefonbuch"?

Asymmetrische Kryptosysteme (2)

- Sinnvolle *Forderungen*:

  1) m lässt sich nicht allein aus $\{m\}_p$ ermitteln

  2) s lässt sich aus p oder einer verschlüsselten, bekannten Nachricht nicht (mit vertretbarem Aufwand) ableiten

  3) $m = \{\{m\}_p\}_s$

  4) evtl. zusätzlich: $m = \{\{m\}_s\}_p$

  (Rolle von Verschlüsselung und Entschlüsselung austauschbar)

- Beachte: "Chosen-Plaintext"-Angriff möglich:
  - beliebige Nachrichten M und deren Verschlüsselung $\{ M \}_p$ jederzeit generierbar, falls p tatsächlich öffentlich
  - das Kryptosystem muss demgegenüber robust sein

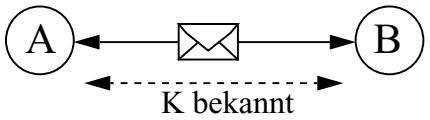
- Vorteil gegenüber symmetrischen Verfahren: vereinfachter Schlüsselaustausch
  - jeder darf den übermittelten public key p mithören
  - secret key s braucht grundsätzlich nie anderen mitgeteilt zu werden
  - bei n Teilnehmern genügen 2n Schlüssel (statt $O(n^2)$ bei sym. Schlüsseln)

- Kenntnis von s *authentifiziert* zugleich den Besitzer
  - "wer $\{ M \}_{pA}$ entschlüsseln kann, der ist wirklich *A*" (wirklich?)

- *Digitale Unterschrift*
  - "wenn (zu M) ein $\{ M \}_{sA}$ existiert mit $\{\{ M \}_{sA} \}_{pA} = M$, dann muss dies (M bzw. $\{ M \}_{sA}$) von A erzeugt worden sein" (wieso?)

sA bzw. pA secret
bzw. public key von A

Authentifizierung mit symmetrischen Schlüsseln



Sei K der zwischen A und B vereinbarte (und geheim-zuhaltende!) Schlüssel

Problem: B soll die Authentizität von A feststellen.

Idee (Geheimdienstprinzip): "Wenn X *das* weiss und kann, dann muss X wirklich X sein, denn sonst weiss und kann das niemand"

Bemerkung: Oft ist eine *gegenseitige* Authentifizierung nötig

1. Verfahren:

A: m := "Ich bin A"
m' := $\{m\}_K$

A→B: m', m ← Damit B den richtigen Schlüssel (für A) wählt

B: überprüfe, ob $\{m\}_K$ = m'

- *Idee*: Überprüfe die Fähigkeit, Nachrichten mit einem geheimen Schlüssel zu kodieren
- *Nachteil*: Möglichkeit von replays durch Abhören

2. Verfahren:

A→B: "Ich bin A"

B→A: n ← Einmalkennung ("nonce")

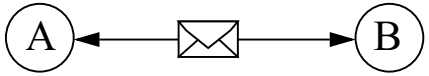
A: n' := $\{n\}_K$

A→B: n'

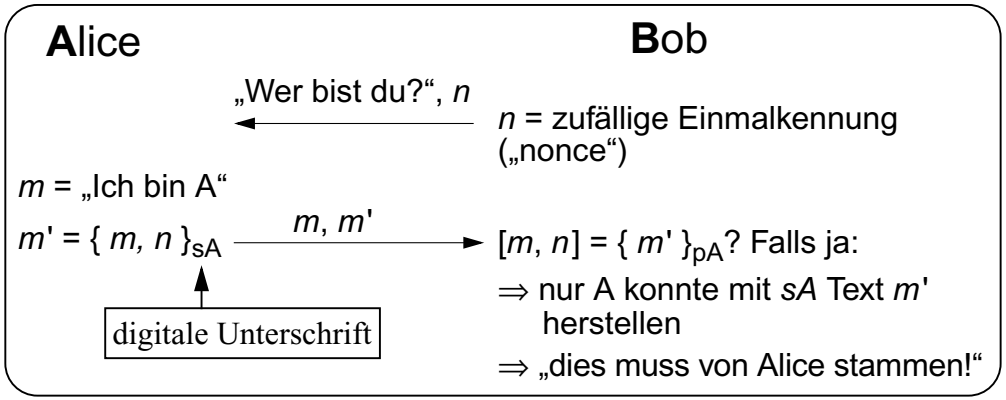
B: überprüfe, ob $\{n\}_K$ = n'

- *Nachteil*: Viele individuelle Schlüsselpaare für jede Client/Server-Beziehung

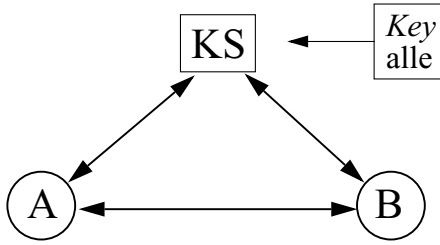
Authentifizierung mit asymmetrischen Schlüsseln



Notation: sX = secret key von X;
pX = public key von X



- Geschützt gegen Replays (wieso?)
- Vorsicht: "Man in the middle"-Angriff möglich (wie?)
- Nachteil: B muss viele public keys speichern; alternativ:



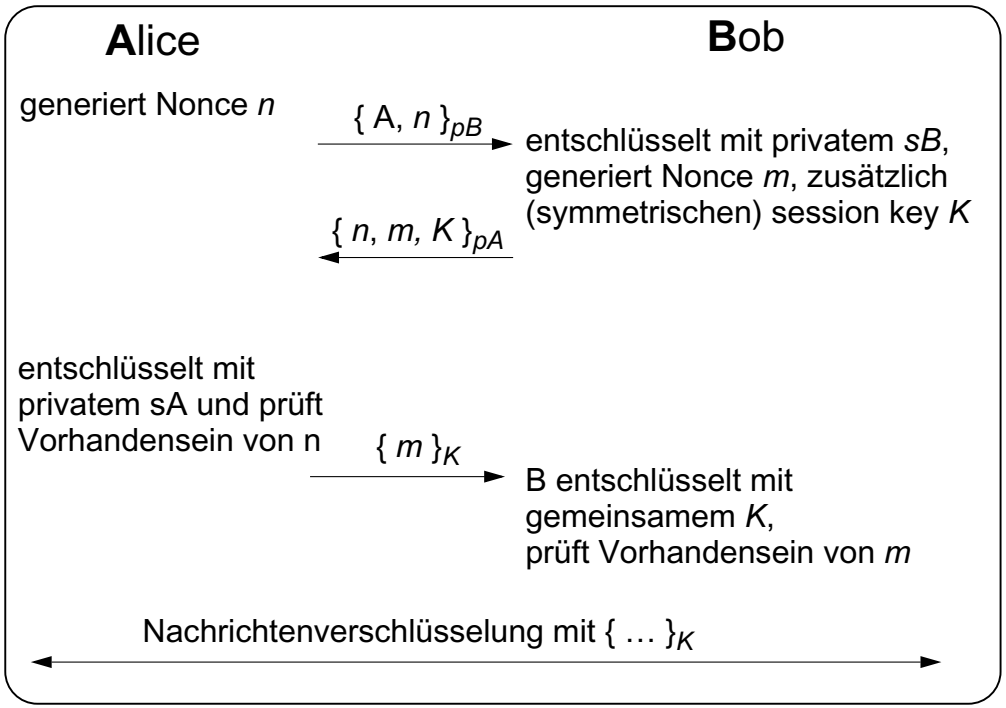
Key Server: Kennt alle public keys

- B erfragt public key von A bei KS
- KS signiert alle seine Nachrichten
- jeder kennt public key von KS (um Unterschrift von KS zu verifizieren)

- Angriff auf den Schlüsselserver KS liefert keine Geheimnisse; erlaubt aber u.U., in dessen Rolle zu schlüpfen und falsche Auskünfte zu geben!
- KS ist evtl. repliziert oder verteilt

Gegenseitige Authentifizierung mit Schlüsselvereinbarung

- Im Prinzip wie oben beschrieben nacheinander in beide Richtungen möglich
- Gleich beides zusammen erledigen ist aber effizienter!
- Hier zusätzlich: Vereinbarung eines symmetrischen "session keys" K, der nach der Authentifizierung zur effizienten Verschlüsselung benutzt wird
- Voraussetzung: A und B kennen die public keys pB bzw. pA des jeweiligen Partners



Replays

- Generelles Problem: Angreifer kann vielleicht eine Nachricht nicht entschlüsseln, jedoch u.U. kopieren und später wieder einspielen
  - elektronische Schecks, Autorisierungscodes für Geldautomaten,...

---

1) Verwendung von *Einmalkennungen*, die vom Empfänger vorgegeben werden ("nonce")

→ (fast) alle Nachrichten sind verschieden
- aufwändiges Protokoll aus mehreren Nachrichten

2) Verwendung von mitkodierten *Sequenznummern*
- nur bei einer Nachrichtenfolge zwischen 2 Prozessen möglich

3) Mitverschlüsseln der *Absendezeit*

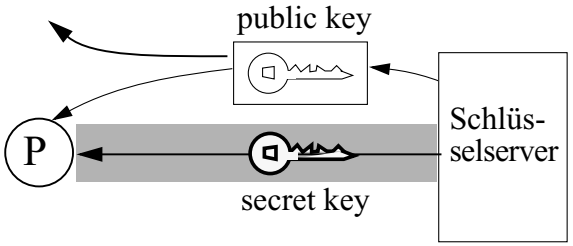
- Empfänger akzeptiert Nachricht nur, wenn seine Zeit max. Δt abweicht.

- lokale Uhrzeit
- globale Zeitapproximation aus Zeitservice (z.B. NTP-Protokoll)
- Empfängerzeit vorher erfragen

- Zeitfenster Δt geschickt wählen!
  - Nachrichtenlaufzeiten berücksichtigen
  - *zu gross* → unsicher durch mögliche Replays
  - *zu klein* → exakte oder häufige Uhrensynchronisation nötig (z.B. vor jeder Nachricht oder nach einem 'reject')
- Angreifer darf Zeitservice nicht manipuliern können!

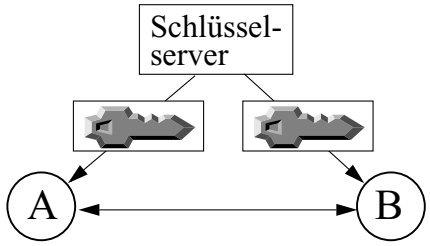
Schlüsselvergabe

- Zur Vergabe eines Paares von public / secret keys:



- secret key muss auf sicherem Kanal zum Client gelangen
- public key von P kann an beliebige Prozesse offen verteilt werden (jedoch i.Allg. “zertifiziert”, dass der Schlüssel authentisch ist)

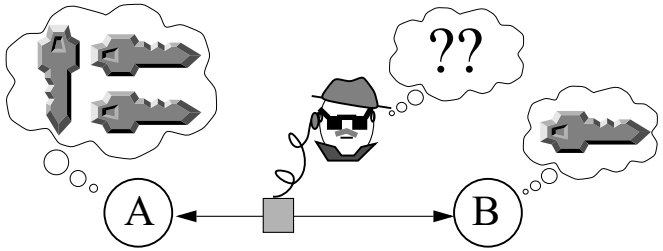
- Zur Generierung von temporären symmetrischen Schlüsseln (“session key”)



Session keys werden sicher und authentisch mit einem Public-Key-Verfahren an zwei Kommunikationspartner übertragen

- Schlüsselserver kann session keys nach Übertragung bei sich löschen
- aufwändiges Public-Key-Verfahren nur ein Mal pro “Session”, tatsächliche Nachrichtenverschlüsselung dann effizient per symm. Schlüssel

Direkte Schlüsselvereinbarung

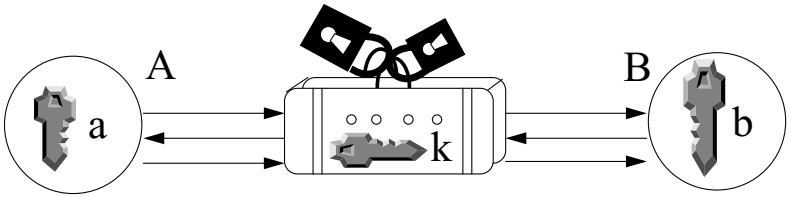


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- Sinnvoll z.B. bei dynamisch gegründeten Prozessen, die vorher noch nie kommuniziert haben
  - z.B. wenn keine public keys vorhanden bzw. nicht bekannt

- Wie geht dies?
  - wir erinnern uns an die “Schatzkiste mit zwei Vorhängeschlössern”

Kommutative Schlüssel



1. A generiert einen Sitzungsschlüssel k
2. A verschlüsselt k mit einem geheimen Schlüssel a
3. A→B: $\{k\}_a$
4. B verschlüsselt dies mit seinem Schlüssel b
5. B→A: $\{\{k\}_a\}_b$
6. A entschlüsselt mit seinem Schlüssel a:
   $\{\{\{k\}_a\}_b\}_{\bar{a}} = \{\{\{k\}_a\}_{\bar{a}}\}_b = \{k\}_b$
7. A→B: $\{k\}_b$
8. B entschlüsselt mit seinem Schlüssel: $\{\{k\}_b\}_{\bar{b}} = k$

a und b sind "lokal erfunden"

Forderung!

Bezeichne $\bar{x}$ den zu x inversen Schlüssel (oft: $\bar{x} = x$)

gemeinsames Geheimnis

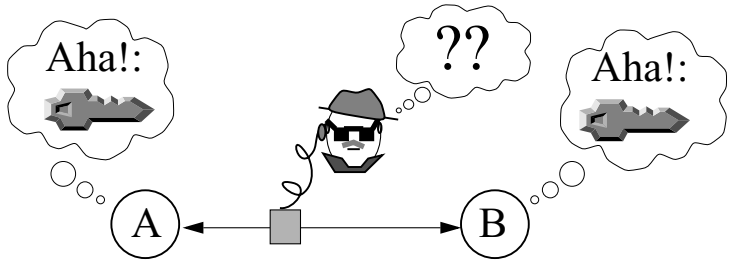
Beachte: k wird nie offen transportiert!

Denkübung: Geht hier *xor* mit "one-time pads" a, b ?

- *xor* erfüllt die Forderung (ist assoziativ und kommutativ)
- *xor* mit one-time pads ist sicher (wirklich?) und effizient
- *Aber:* Wenn Schritt 3 ($\{k\}_a$) und Schritt 5 ($\{\{k\}_a\}_b$) abgehört wird, dann kann daraus der Schlüssel b ermittelt werden, so dass aus dem abgehörten Schritt 7 ($\{k\}_b$) das geheime k ermittelt werden kann!
- Gibt es anstelle von *xor* andere (sichere!) Verschlüsselungsoperationen?

Schlüsselvereinbarung mit Diffie-Hellman-Verfahren

Ziel: A und B sollen sich über einen unsicheren Kanal auf ein gemeinsames "Geheimnis" G einigen, ohne dass ein Angreifer es erfährt



- Nutzung einer *Einwegfunktion*: $f(x) = c^x \bmod p$

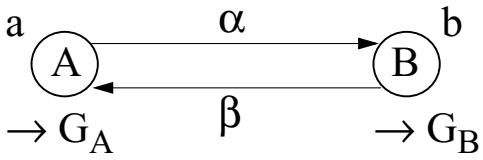
  $(1 < c < p$; i.Allg. ist p eine grosse Primzahl)

  - in einem Restklassenring ist die Bestimmung *diskreter Logarithmen* (und k-ter Wurzeln) wesentlich schwieriger als die Bildung von Potenzen

  - im RPC-Protokoll von Sun wurde z.B. c = 3 gewählt und p = d4a0ba0250b6fd2ec626e7efd637df76c716e22d0944b88b (hex.); eine Zahl aus 192 Bits (die Parameter c und p sind kein Geheimnis)

  - gelegentlich wird für p auch ein Produkt aus zwei grossen Primzahlen empfohlen, oder es wird $p = 2^n$ gewählt, da dann die mod-Operation besonders einfach zu berechnen ist

Der Diffie-Hellman-Algorithmus



- wenig Nachrichten
- effizient

1. A wählt eine Zufallszahl a
2. A berechnet $\alpha = f(a)$
3. A → B: α
4. B wählt eine Zufallszahl b
5. B berechnet $\beta = f(b)$
6. B → A: β
7. A berechnet $G_A = \beta^a \bmod p$
8. B berechnet $G_B = \alpha^b \bmod p$

(a und b sind nur lokal bekannt und bleiben geheim)

Behauptung: $G_A = G_B$ (gemeinsames Geheimnis!)

Beispiel (für c = 5 und unrealistisch kleines p = 7):

$f(x) = 5^x \bmod 7$

$a = 3 \rightarrow \alpha = 6$ $\quad \rightarrow G_B = 6^4 \bmod 7 = 1$

$b = 4 \rightarrow \beta = 2$ $\quad \rightarrow G_A = 2^3 \bmod 7 = 1$

$G_A = G_B$

Zu zeigen: $\beta^a \bmod p = \alpha^b \bmod p$, also:

$$(c^b \bmod p)^a \bmod p = (c^a \bmod p)^b \bmod p$$

Lemma: $(k \bmod p)^n \bmod p = k^n \bmod p$ ← Restklassen-arithmetik...

$$\begin{aligned}
\mathbf{(c^b \bmod p)^a \bmod p} &= (c^b)^a \bmod p \quad \text{[Lemma]} \\
&= c^{(b\,a)} \bmod p \\
&= c^{(a\,b)} \bmod p \\
&= (c^a)^b \bmod p \quad \text{[Lemma]} \\
&= \mathbf{(c^a \bmod p)^b \bmod p}
\end{aligned}$$

Bemerkungen:

- Lässt sich auch auf k > 2 Benutzer verallgemeinern
- Der Algorithmus (entdeckt 1976) ist patentiert
  - U.S.-Patent Nummer 4200770 (Sept. 1977)

United States Patent [19]

Hellman et al.

[11] 4,218,582

[45] Aug. 19, 1980

[54] PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD

[75] Inventors: Martin E. Hellman, Stanford; Ralph C. Merkle, Palo Alto, both of Calif.

[73] Assignee: The Board of Trustees of the Leland Stanford Junior University, Stanford, Calif.

[21] Appl. No.: 839,939

[22] Filed: Oct. 6, 1977

[51] Int. Cl.[2] H04L 9/04

[52] U.S. Cl. 178/22; 364/900

[58] Field of Search 178/22

[56] References Cited

PUBLICATIONS

"New Directions in Cryptography," Diffie et al., *IEEE Transactions on Information Theory*, vol. II22, No. 6, Nov. 1976, pp. 644–654.

"A User Authentication Scheme not Requiring Secrecy in the Computer," Evans, Jr., et al., *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 437–442.

"A High Security Log-In Procedure," Purdy, *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 442–445.

Diffie et al., "Multi-User Cryptographic Techniques," *AFIPS Conference Proceedings* vol. 45, pp. 109–112, Jun. 8, 1976.

Primary Examiner—Howard A. Birmiel

[57] ABSTRACT

A cryptographic system transmits a computationally secure cryptogram that is generated from a publicly known transformation of the message sent by the transmitter; the cryptogram is again transformed by the authorized receiver using a secret reciprocal transformation to reproduce the message sent. The authorized receiver's transformation is known only by the authorized receiver and is used to generate the transmitter's transformation that is made publicly known. The publicly known transformation uses operations that are easily performed but extremely difficult to invert. It is infeasible for an unauthorized receiver to invert the publicly known transformation or duplicate the authorized receiver's secret transformation to obtain the message sent.

17 Claims, 13 Drawing Figures

US4218582: Public key cryptographic apparatus and method

Inventors: Martin E. Hellman, Stanford; Ralph C. Merkle, Palo Alto

Issued/Filed Dates: Aug. 19, 1980 / Oct. 6, 1977

Abstract:

A cryptographic system transmits a computationally secure cryptogram that is generated from a publicly known transformation of the message sent by the transmitter; the cryptogram is again transformed by the authorized receiver using a secret reciprocal transformation to reproduce the message sent. The authorized receiver's transformation is known only by the authorized receiver and is used to generate the transmitter's transformation that is made publicly known. The publicly known transformation uses operations that are easily performed but extremely difficult to invert. It is infeasible for an unauthorized receiver to invert the publicly known transformation or duplicate the authorized receiver's secret transformation to obtain the message sent.

What is claimed is:

1. In a method of communicating securely over an insecure communication channel of the type which communicates a message from a transmitter to a receiver, the improvement characterized by: providing random numbers at the receiver; generating from said random numbers a public enciphering key at the receiver; generating from said random numbers a secret deciphering key at the receiver such that the secret deciphering key is directly related to and computationally infeasible to generate from the public enciphering key; communicating the public enciphering key from the receiver to the transmitter; processing the message and the public enciphering key at the transmitter and generating an enciphered message by an enciphering transformation, such that the enciphering transformation is easy to effect but computationally infeasible to invert without the secret deciphering key; transmitting the enciphered message from the transmitter to the receiver; and processing the enciphered message and the secret deciphering key at the receiver to transform the enciphered message with the secret deciphering key to generate the message.

2. ...

...

17. ...

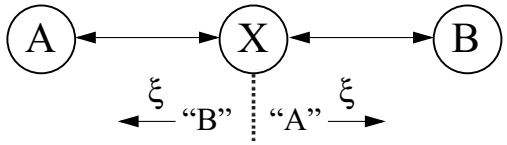
Sweet Little Secret G

- A und B könnten $G = G_A = G_B$ als symmetrischen Schlüssel zur Kodierung ihrer Nachrichten verwenden

- *Besser*: G nur als Schlüssel verwenden, um einen zufällig bestimmten session key zu kodieren und dem Kommunikationspartner diesen mitzuteilen
  - so wird es z.B. im Sun-RPC-Protokoll gemacht
  - Motivation: G selbst so selten wie möglich benutzen

- Einzusehen bliebe noch, dass aus Kenntnis von α und β (sowie von c und p aus f) G von einem passiven Angreifer nicht (effizient) ermittelt werden kann!
  - $\alpha = c^a \bmod p \rightarrow$ a ist ein *diskreter Logarithmus*; dieser ist i.Allg. "schwierig" zu berechnen!
  - Bem.: nicht jedes p ist "gut"; sollte auch einige 100 Bit gross sein
  - "Probieren" aller a, bis $\alpha = c^a \bmod p$ gefunden $\rightarrow$ zu langwierig
  - α und β sind *unabhängig* voneinander! (Wieso ist das ein Argument?)

- Wie ist es aber bei *aktiven Angreifern*?

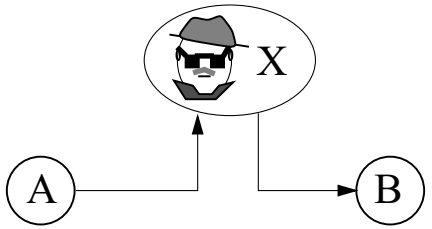


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

- X kann unter Vortäuschung falscher Identitäten jeweils eigene Schlüssel für die Teilstrecken AX und XB vereinbaren!

Aktive Angriffe durch Eindringen und Schlüsselfälschung

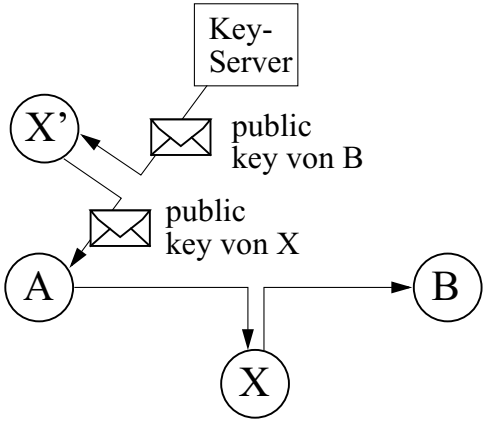
Szenario 1:



- X verhält sich gegenüber A wie B, gegenüber B wie A ($\rightarrow$ X arbeitet "transparent")
- z.B. eigene Schlüssel für die Teilstrecken vereinbaren

- Challenge-Response-Test nützt so nichts: X reicht Challenges einfach an den von ihm vorgetäuschten Partner weiter und miemt mit der abgefangenen Antwort die angenommene Identität

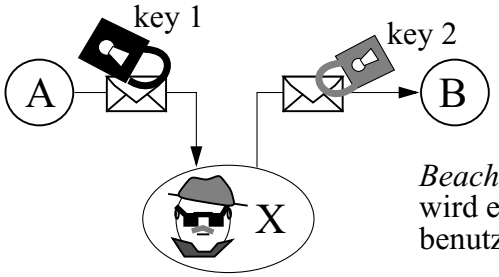
Szenario 2:



- kompromittierter Key-Server; Verschwörung,...
- X kann alle von A mit dem falschen Schlüssel verschlüsselten Nachrichten an B entziffern
- X verschlüsselt danach die Nachricht mit dem richtigen Schlüssel für B

- digitale Unterschrift des Key-Servers nützt nichts, wenn A den Prozess X' für den Key-Server hält und dessen Unterschrift akzeptiert!
- nützt die allgemeine Bekanntgabe des public keys des Key-Servers?
- ist es überhaupt möglich, X in diesen Szenarien zu erkennen?

Erkennen von Eindringlingen



Beachte: Auf der Strecke AX wird ein anderer Schlüssel benutzt, als auf der Strecke XB!

1) B stellt eine Anfrage, die nur A beantworten kann
2) A generiert die Antwort und verschlüsselt diese
3) A sendet zunächst nur die "Hälfte" davon zurück
   - z.B. nur jedes zweite Bit (die "geraden" Bits)
   - B erwartet diese Hälfte der Antwort in weniger als t Zeiteinheiten
4) Ohne die andere Hälfte kann X diese nicht entschlüsseln und neu verschlüsseln
   (sofern X nicht erzwingen kann, dass key 1 = key 2 ist)
5) Erst nach t Zeiteinheiten sendet A die andere Hälfte
   - B setzt Schlüsseltexthälften zusammen und überprüft Antwort

---

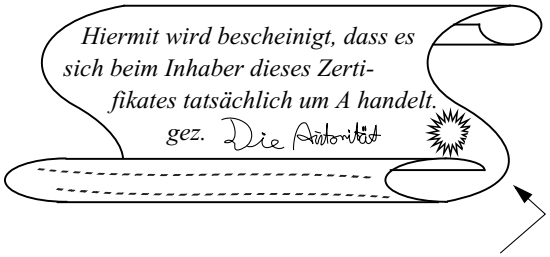
→ Gibt X die halbe Nachricht unverändert weiter, kann B diese nicht entschlüsseln → *Fälschung erkannt*

→ Behält X die halbe Nachricht bis zum Eintreffen der anderen Hälfte (und speichert die andere Hälfte dann t Zeiteinheiten zwischen), dann arbeitet X nicht mehr zeittransparent → *Eindringling erkannt*

Frage: Wird in 1) nicht schon ein gemeinsames Geheimnis vorausgesetzt? Können (im Kontext des Diffie-Hellman-Verfahrens) A und B nicht dieses benutzen, um einen von X nicht ermittelbaren gemeinsamen Schlüssel zu finden? Oder genügt in 1) eine schwächere Eigenschaft ("originelle" Antwort; Fähigkeit, die nur A hat...)?

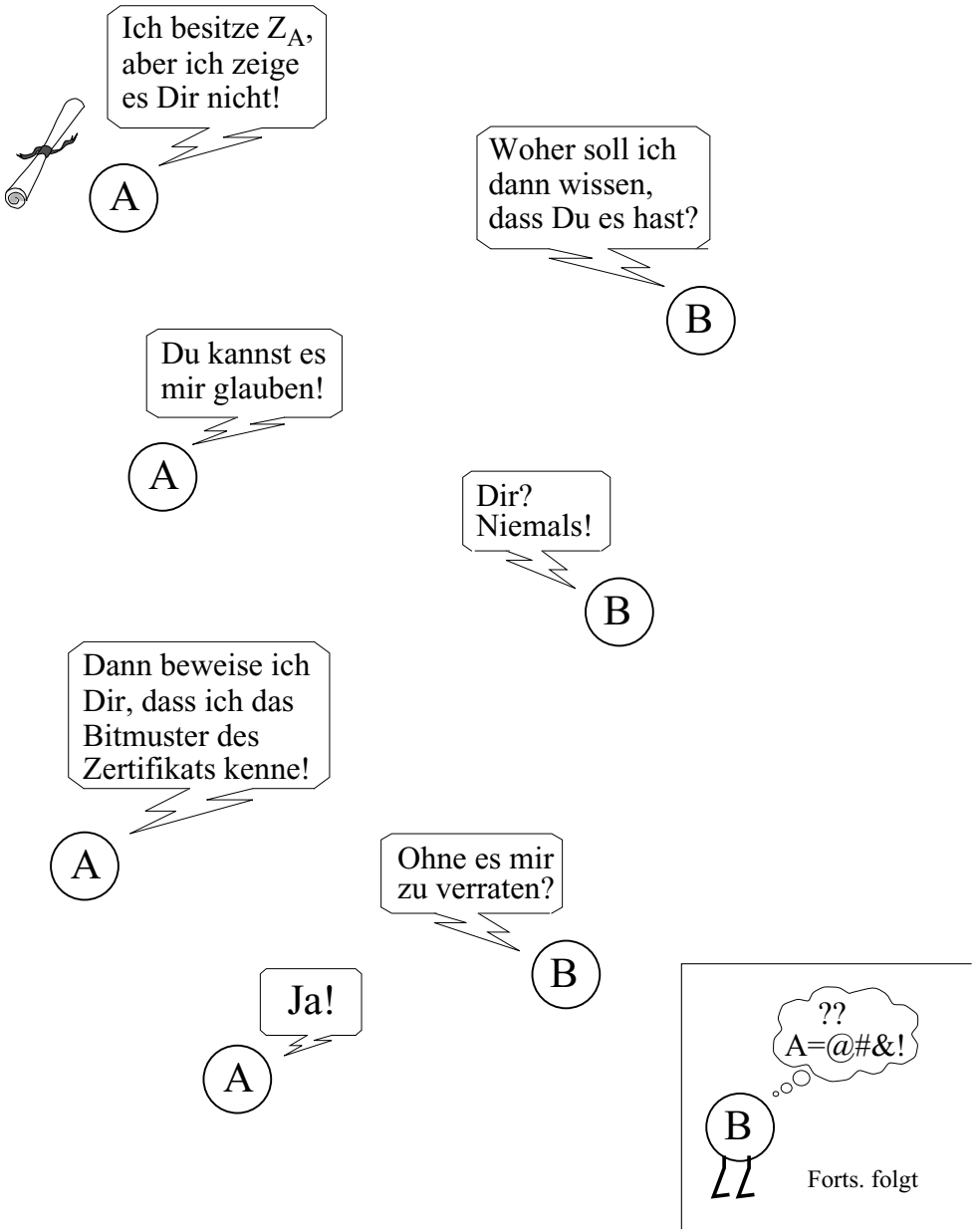
Authentifizierung mit Zertifikaten?

- Die Idee:



- A lässt sich einmalig von einer Autorität ein *Zertifikat* Z_A mitgeben (sollte von der Autorität signiert sein)
  - Autorität gilt als vertrauenswürdig und hat A evtl. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)
- Wenn B an der Identität von A zweifelt, wird B von A auf sein Zertifikat Z_A hingewiesen
  - Besitz des Zertifikates = Authentifizierung
- Aber: A darf Z_A nie B zeigen - sonst könnte B es sich kopieren und sich fortan als A ausgeben!
  - wie vermeidet man "raubkopierte Zertifikate"?
  - in der digitalen Welt lassen sich Bitfolgen perfekt kopieren (im Unterschied zu klassischen "fälschungssicheren" Ausweisen)
- Z_A muss offenbar ein *Geheimnis* bleiben, das ausser der Autorität und A niemand kennt!
- Taugt ein solches Geheimnis als Zertifikat??
  - wie beweist man den Besitz eines Zertifikates, ohne es zu zeigen?

Geheime Zertifikate?



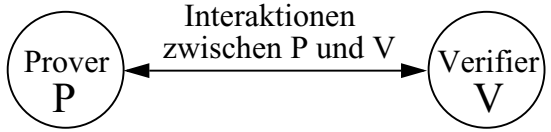
Geheime Zertifikate!

- Im Prinzip wissen wir schon, dass das geht: Der secret key sA eines asymmetrischen Verfahrens stellt ein solches Zertifikat dar
  - braucht von A nicht verraten zu werden
  - B kann dennoch überprüfen, ob A das Zertifikat hat (z.B. indem sich B von A etwas mit sA verschlüsseln lässt und anschliessend durch Anwenden von pA prüft; oder indem B ein $\{M\}_{pA}$ an A schickt und sich dies von A mit sA entschlüsseln lässt)

- Eine andere Realisierung geht mit “zero knowledge”
  - beweist Kenntnis eines Geheimnisses G, ohne relevante Information preiszugeben

Zero-Knowledge-Beweis

- “Beweis” = Nachweis, dass P eine bestimmte Folge von Bits (= Zahl, Algorithmus, Zertifikat,...) kennt



Hat Kenntnis einer bestimmten (“geheimen”) Information

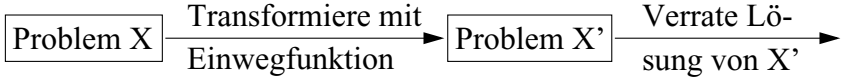
Überprüft, ob P diese Kenntnis wirklich hat

- P kann V (praktisch) *nicht betrügen:* Wenn P die Information nicht hat, sind seine Chancen, V zu überzeugen, verschwindend gering

- V *erfährt nichts* über die eigentliche Kenntnis von P
  - V erfährt auch sonst nichts Relevantes von P, was V nicht auch alleine in Erfahrung bringen könnte

---

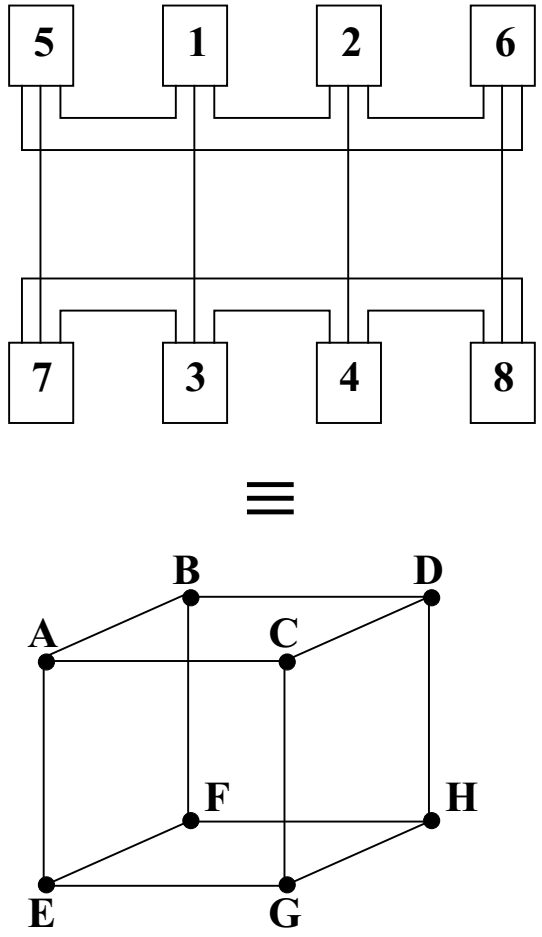
Idee:



(Wobei die Lösung von X’ die Lösung von X logisch impliziert, sie jedoch nicht effektiv-konstruktiv liefert)

Beispiel: Isomorphie von Graphen

Bemerkung: Ob zwei grosse (z.B. in Form von Adjazenzmatrizen) gegebene Graphen G_1, G_2 topologisch isomorph ($G_1 \sim G_2$) sind (d.h. bis auf Umbenennung von Knoten und evtl. Kanten identisch sind), ist ein *schwieriges* Problem.



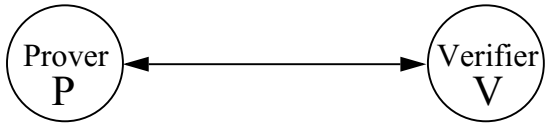
Hier nur ein kleines und daher einfaches (also unrealistisches) Beispiel

A = 7
B = 5
C = 8
D = 6
E = 3
F = 1
G = 4
H = 2

Überprüfung eines durch eine Knotenzuordnung gegebenen Isomorphismus ist allerdings “einfach”!

Zero-Knowledge mit Graphisomorphie

- P behauptet, einen Beweis zu haben, dass zwei gegebene Graphen G_1, G_2 isomorph sind, möchte den Beweis aber nicht verraten



- Folgendes Protokoll *überzeugt* V davon:
  - P erzeugt durch zufällige Permutation der Knoten einen Graphen H mit $H \sim G_1$ (und damit $H \sim G_2$). Für P ist dies einfach. Andere aber können $H \sim G_1$ oder $H \sim G_2$ nicht einfacher entscheiden als $G_1 \sim G_2$
  - P sendet H an V
  - Entweder bittet V dann P
    - $H \sim G_1$ nachzuweisen, *oder*
    - $H \sim G_2$ nachzuweisen

    zufällig; bzw. von P nicht vorhersehbar
  - Da P den Graphen H konstruiert hat, kann P das gewünschte einfach tun (P hütet sich jedoch davor, auch noch die andere, von V nicht gewünschte, Alternative nachzuweisen - wieso?)
  - P und V wiederholen alles n Mal, wobei von P jedesmal ein anderer "Zeuge" H konstruiert wird (Beweissicherheit = $1\text{-}2^{-n}$)

- Der Isomorphismus bleibt dabei ein Geheimnis von P!

Zero-Knowledge: Eigenschaften

- Falls P *keinen* Isomorphismus zwischen G_1 und G_2 kennt (also *lügt*), kann P keinen Graphen H konstruieren, der nachweislich isomorph zu *beiden* ist
  - *verschiedene* H_1, H_2 zu finden mit $H_1 \sim G_1$ und $H_2 \sim G_2$ ist einfach; mit 50% Wahrscheinlichkeit wird P dann allerdings der Lüge überführt!
- V *lernt nichts* über die Isomorphie $G_1 \sim G_2$, *glaubt* aber schliesslich, dass P eine solche kennt
- Zur Minimierung der Interaktionen lassen sich die "Runden" *parallelisieren*: P sendet *mehrere* "isomorphe Zeugen" an V, und V sendet einen Bitvektor zurück, der die Einzelnachweise auswählt
- V kann einem Dritten W gegenüber nicht beweisen, dass P den Isomorphismus kennt: Selbst ein exaktes Protokoll der Kommunikationsvorgänge muss W nicht überzeugen: P und V könnten sich *verschworen* haben!
- Da V nichts Relevantes gelernt hat, kann V sich anderen gegenüber auch nicht mit der Kenntnis schmücken
  - sich also *nicht für P ausgeben* (wenn die Kenntnis P identifiziert)

Grosse Graphen sind in der Praxis etwas unhandlich. Es gibt praktischere Ausprägungen des Zero-Knowledge-Verfahrens, z.B. das Protokoll von Fiat und Shamir. Dieses beruht auf der Schwierigkeit, die k-te Wurzel in einem Restklassenring zu berechnen.

Der Kerberos-Sicherheitsdienst

- Protokoll zur Schlüsselvergabe, Authentifizierung und Einrichtung sicherer Kommunikationskanäle
- Am MIT entwickelt im Rahmen eines erstes grossen Client-Server-Campusnetzes
  - war dort ab 1986 im Einsatz
- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln (z.B. DES)
- Public domain; es gibt auch kommerzielle Varianten
- In heutigen "offenen" verteilten Systemen gibt es noch andere Systemdienste zur Erhöhung der Sicherheit
  - z.B. ssh, VPN etc.

R.M. Needham, M.D. Schroeder: *Using Encryption for Authentication in Large Networks of Computers.* CACM 21(12), pp. 993-999, 1978

B. Clifford Neuman and Theodore Ts'o: *Kerberos: An Authentication Service for Computer Networks.* IEEE Communications Magazine, Volume 32, Number 9, pp. 33-38, September 1994

RFC 1510: *The Kerberos Network Authentication Service (V5),* www.rfc-archive.org/getrfc.php?rfc=1510

Vgl. auch *Wikipedia*

Kerberos-Prinzipien

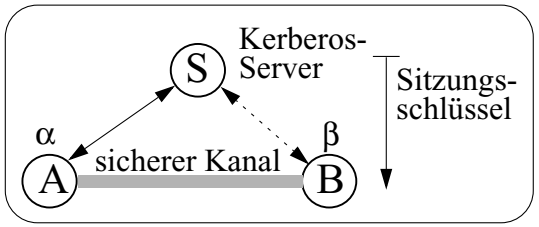
- Offenes Campusnetz → Nachrichten prinzipiell unsicher
- Kommunikation daher i.Allg. verschlüsselt und nur mit authentifizierten Partnern
  - Kenntnis des Sitzungsschlüssels als Authentitätsbeweis
- Passwörter niemals im Klartext übertragen
  - auch keine Passwortspeicherung

"Kerberos-Server"

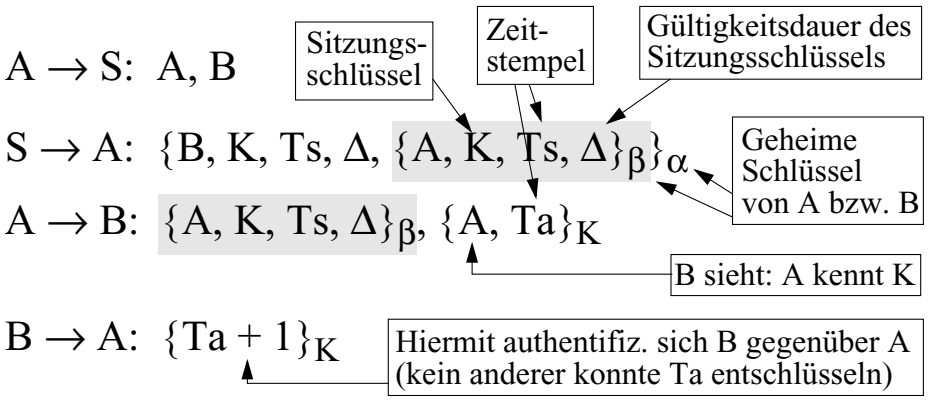
- Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: "KDC") akkreditiert
  - vereinbaren mit dem KDC auch ihren Geheimschlüssel ("master key")
  - ohne Akkreditierung keine Server-Berechtigungsscheine ("Tickets")
  - ohne Tickets kein Service
  - Ticket nur in Verbindung mit Authentitätsnachweis gültig
- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet
- Mehrere Sicherheitsstufen möglich, z.B.:
  (1) Authentifizierung nur bei Einrichtung eines Kommunikationskanals
  (2) Authentifizierung bei jeder Nachricht zwischen A und B
  (3) zusätzlich Verschlüsselung der Nachrichten

Kerberos-Anwendungsbeispiel: Einrichtung eines sicheren Kanals

- Gegenseitige Authentifizierung (via Kerberos Server)
- Verwendung eines Sitzungsschlüssels ("session key")
- $\{X, K, Ts, \Delta\}_\gamma$ heisst "Ticket"
  - Tickets kann man an andere ("vertrauenswürdige") Instanzen weitergeben

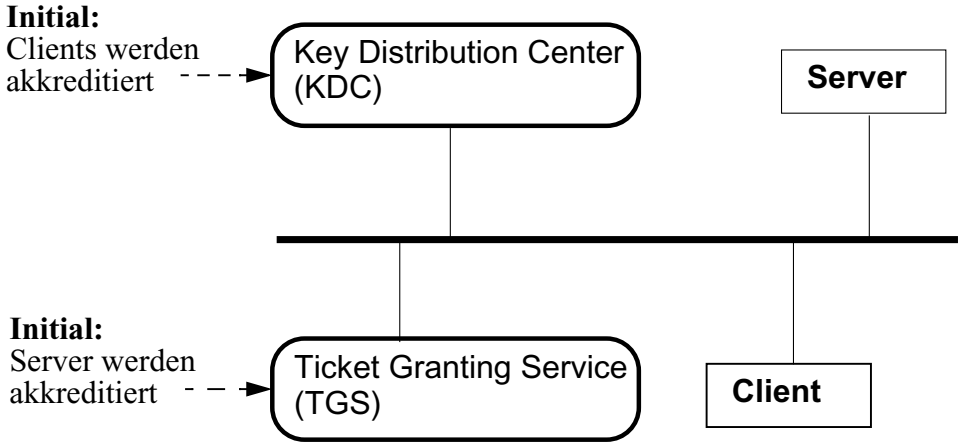


Hier: Version 4; andere Versionen im Prinzip nur leicht unterschiedlich



- Geheimschlüssel α von A und β von B darf niemand ausser S kennen! (Kenntnis wird als Identitätsnachweis betrachtet)
- A reicht hier ein von S erhaltenes (mit β codiertes) Ticket an B weiter

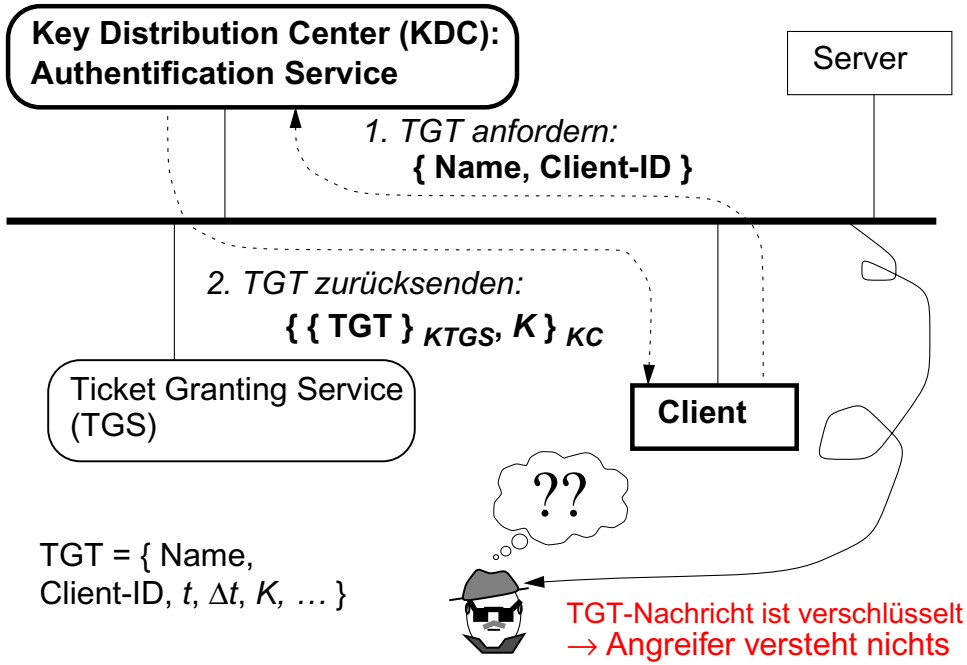
Kerberos: Akkreditierung



- Benutzer (Clients) und deren Passwörter (= Schlüssel) werden dem KDC bekannt gemacht
- TGS und dessen geheimer Schlüssel werden ebenfalls beim KDC akkreditiert
- Server und deren geheime Schlüssel werden dem TGS bekannt gemacht
  - es kann mehrere TGS-Server geben (→ Lastverteilung)

Kerberos: TGT-Anforderung

- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



- Client an KDC: sendet { Name, Client-ID } im Klartext
- KDC: wählt K; erstellt TGT = { Name, Client-ID, t, Δt, K, … }
- KDC an Client: sendet { { TGT } $_{KTGS}$, K } $_{KC}$ zurück;
  KC = h(Passwort); KTGS = TGS-Schlüssel; K = Sitzungsschlüssel
- Client: gewinnt { TGT } $_{KTGS}$ und K durch Entschlüsselung mit Passwort:
  - (chiffriertes) TGT berechtigt zum Erwerb von Service Tickets;
  - K sichert Kommunikation mit TGS gegen Angreifer

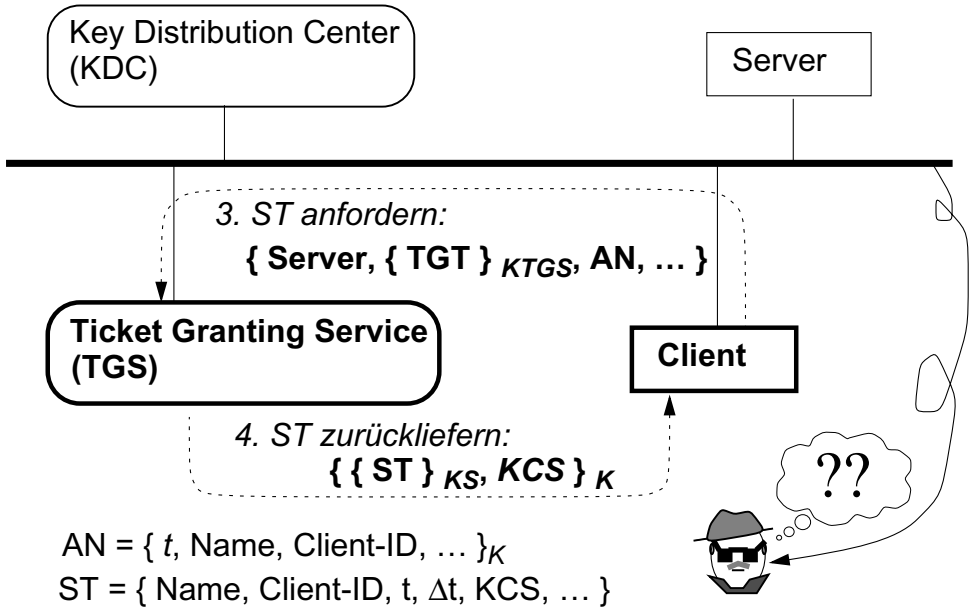
→ KDC-Nachricht ist authentisch: Nur KDC kennt noch Schlüssel KC !

→ Nur der echte Client kann TGT mittels KC nutzbar machen

→ Passwort verlässt Client-Rechner nicht

→ TGT ist verschlüsselt, nur für Zeitspanne Δt gültig, geht nur an Client

Kerberos: Service Ticket erwerben

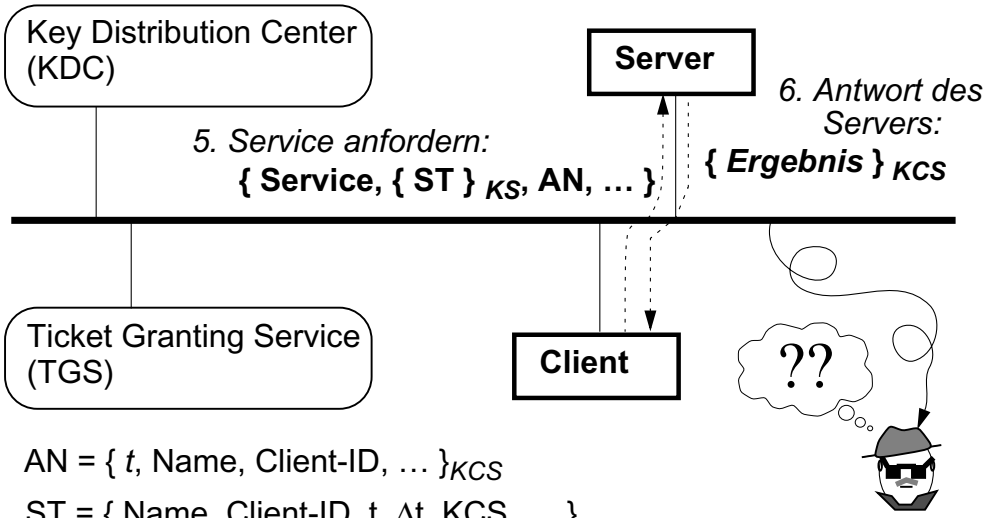


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } $_{K}$
- Client sendet an TGS { Server, { TGT } $_{KTGS}$, AN, … } als Request
- TGS: entschlüsselt TGT mit Schlüssel KTGS, erhält damit K;
  entschlüsselt AN mit K, vergleicht Inhalt mit TGT;
  erstellt Service Ticket ST = { Name, Client-ID, t, Δt, KCS, … }
- TGS sendet an Client { { ST } $_{KS}$, KCS } $_{K}$ zurück
- Client: gewinnt { ST } $_{KS}$ und KCS durch Entschlüsselung mit K:
  - (chiffriertes) ST berechtigt zur Nutzung des Servers
  - KCS sichert Kommunikation zwischen Client und Server

→ Ohne Sitzungsschlüssel K ist ST nicht nutzbar: Nur Client kennt K !

→ ST höchstens für Zeitspanne Δt gültig

Kerberos: Nutzung des Service

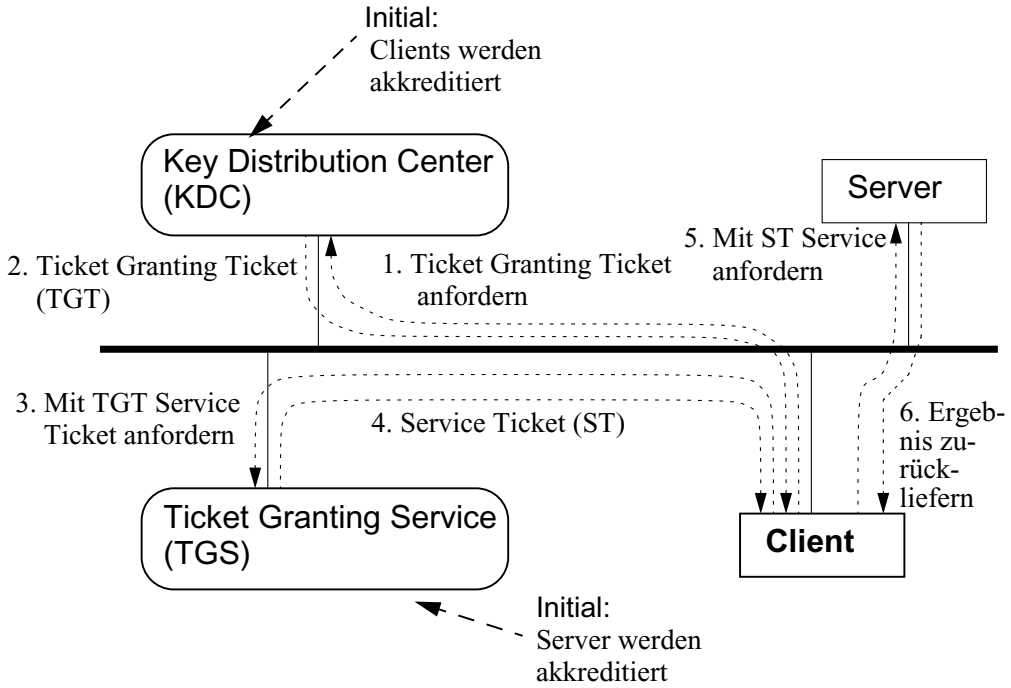


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } KCS
- Client an Server: sendet { Service, { ST } KS, AN, … } als Service-Request
- Server: entschlüsselt ST mit KS, erhält damit KCS;
  entschlüsselt AN mit KCS, vergleicht Inhalt mit ST;
  leistet Service und erzeugt Ergebnisdaten
- Server an Client: antwortet mit { Ergebnisdaten } KCS
- Client: authentifiziert und entschlüsselt das Ergebnis mittels KCS

→ Folgedialoge zwischen Client und Server mittels KCS verschlüsselbar

→ ST als Einmal-Ticket oder evtl. innerhalb Δt mehrfach nutzbar

Kerberos: Protokollübersicht



- Protokoll ist zweistufig:
  - Client kommuniziert nur selten mit dem KDC (1,2) → eigentlicher Geheimschlüssel (Passwort-basiert) wird nur selten benutzt
  - ein TGT ist für mehrere Anfragen beim Ticket-Service gültig

Kerberos - weitere Aspekte

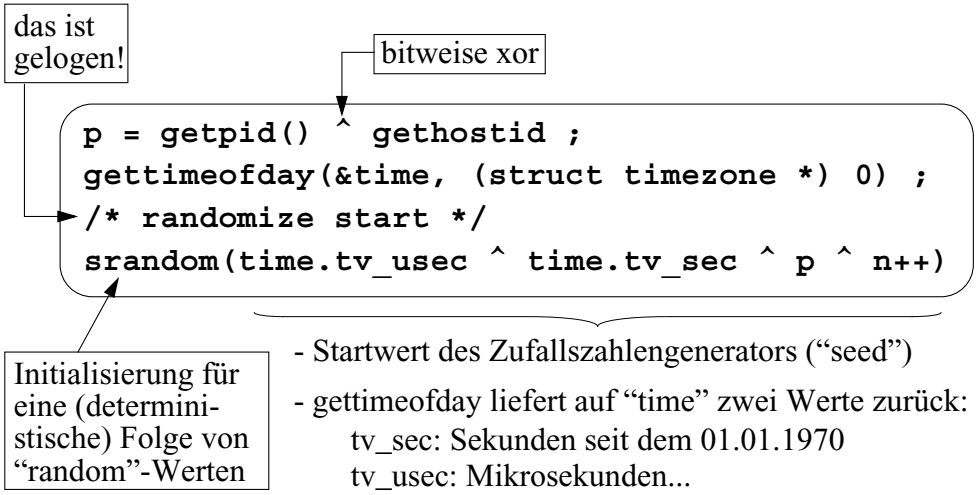
- Nachrichten enthalten noch weitere (technische) Angaben
  - z.B. Versionsnummer, Nachrichtentyp, Prüfsumme, Netzwerkadresse,...
- Es gibt dezentrale Zuständigkeitsbereiche (“realms”)
  - lok. KDC vermittelt Zugangsticket zu KDC eines fremden Bereichs
- Kerberos-Software enthält u.a.:
  - Library mit Routinen, um Authentifizierungsanforderungen erzeugen und lesen zu können, Nachrichten zu authentifizieren und zu verschlüsseln
  - Datenbank und Verwaltungsroutinen für registrierte Nutzer (Geheimschlüssel, Gültigkeitsdauer, Verwaltungsdaten,...)
  - Tools zur Replikation der Datenbank (Verteilung ist wichtig, da bei Ausfall des KDC im ganzen Netz fast nichts mehr geht!)
- Neuere Versionen (gegenüber Version 4): mehr Funktionalität und allgemeiner verwendbar, z.B.:
  - standardisierte Datenformate
  - Verbesserung einiger Sicherheitskonzepte; Alternativen zu DES
  - besser skalierbare Authentifikation über fremde Zuständigkeitsbereiche
  - Unterstützung erneuerbarer und transferierbarer Tickets
- Weiterentwicklungen
  - z.B. asymm. Schlüssel, Einbindung von Chipkarten, verteilte Datenbank,...
- Kerberos ist weit verbreitet (“Quasi-Standard”)
  - z.B. um verteilte Dateiserver zu sichern oder modifizierte Versionen von telnet, rlogin, rcp, rsh, ftp etc. zu ermöglichen
  - kommerzielle Varianten z.T. nicht kompatibel zueinander
  - Microsoft: Unterstützung ab Windows 2000

Kerberos - Sicherheitsaspekte

- KDC und TGS müssen geschützt werden
  - z.B. gegen unbefugtes Lesen der Datenbank, Verändern der Daten, denial of service,...
- Tickets sollen vom Client in einem “sicheren Speicherbereich” aufbewahrt werden
  - Master key (aus Passworteingabe des Benutzers abgeleitet) wird sobald wie möglich aus dem Speicher gelöscht
- Uhren der Kommunikationspartner und der Kerberos-Server müssen “verlässlich” synchronisiert werden
  - innerhalb eines gewissen Toleranzintervalls von einigen Minuten
  - Störung des Uhrenabgleichs erlaubt evtl. mehrfachen Ticketmissbrauch
- Replays sind innerhalb der Gültigkeitsdauer (typw.: einige Minuten bis Stunden) prinzipiell möglich!
  - Server sollte alte, noch gültige Tickets speichern, um Replays erkennen zu können
- Auf Public-Domain-Servern (und CDs etc.) könnte gefälschte Software vorhanden sein (“trojanische Pferde”)
- “Erster” Schlüssel basiert auf einem Passwort → Off-line-Attacke durch Raten gängiger Passworte
- Hintertüren und Angriffe ausserhalb von Kerberos
  - fremde Tickets lesen (Netz-sniffer, Superuser-Rechte beschaffen,...)
  - “Hijacking” von TCP-Verbindungen

Schlüsselgenerierung

- Die Schlüsselgenerierung von Kerberos (z.B. für TGT oder Sitzungsschlüssel) funktionierte ursprünglich so:



- Lässt sich aus einem Schlüssel der folgende berechnen?
  - p ist eine "Konstante"
  - time of day ist (ungefähr) bekannt
  - n++ unterscheidet sich i.Allg. nur in wenigen Bits von n

- Könnte jemand vielleicht absichtlich die Uhr des Servers auf einen falschen (d.h. bekannten) Wert setzen?
  - welche Granularität hat eigentlich die Uhr?

- Der Algorithmus ist inzwischen "verbessert"...

- Und die Moral der Geschichte?